# Python 代码整理汇总-2



# Import Data

## Read in data from TXT file

```
FileName = 'cars.txt'
fid = open(FileName, 'r');
print(fid.read())
fid.close() ## 很重要，记得关
```

```
Call me Ishmael. Some years ago--never mind how long precisely--having
    little or no money in my purse, and nothing particular to interest me on
    shore, I thought I would sail about a little and see the watery part of
    the world. It is a way I have of driving off the spleen and regulating
    the circulation. Whenever I find myself growing grim about the mouth;
    whenever it is a damp, drizzly November in my soul;
```

# Read file using iter

```
FileName = 'cars.txt'
fid = open(FileName, 'r');
get_line = iter(fid)
print(next(get_line), end="")
print(next(get_line), end="")

print('-'*50)
for line in iter(fid):
    print(line, end="")
```

```
Call me Ishmael. Some years ago--never mind how long precisely--having
    little or no money in my purse, and nothing particular to interest me on
--------------------------------------------------
    shore, I thought I would sail about a little and see the watery part of
    the world. It is a way I have of driving off the spleen and regulating
    the circulation. Whenever I find myself growing grim about the mouth;
    whenever it is a damp, drizzly November in my soul;
```

## with open

```
with open(FileName, 'r') as fid: ## 强烈建议去用，自动关掉文件
    print(fid.readline())
    print(fid.readline())
```

```
Call me Ishmael. Some years ago--never mind how long precisely--having

    little or no money in my purse, and nothing particular to interest me on
```

# Numpy for reading csv file

ref: data = np.loadtxt(filename, delimiter='\t', skiprows=1, usecols=[0,2], dtype=float)

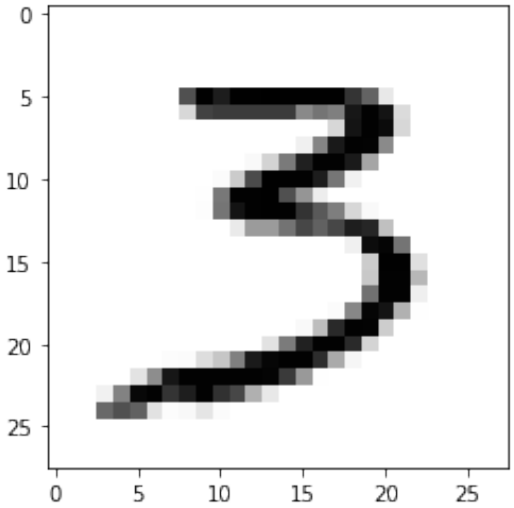
```
import numpy as np
import matplotlib.pyplot as plt
file = 'digits.csv'

digits = np.loadtxt(file, delimiter=',')
print(type(digits),digits.shape)

# Select and reshape a row
im = digits[25, 1:]
im_sq = np.reshape(im, (28, 28)) # 1+28+28 =785
```

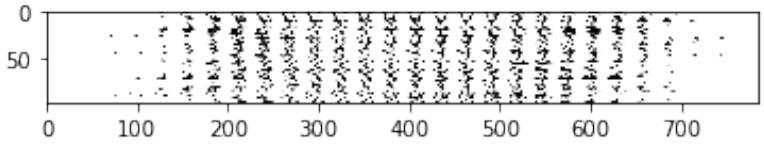
```
11
12 # Plot reshaped data
13 # imshow函数
14 plt.imshow(im_sq, cmap='Greys', interpolation='nearest')
15 plt.show()
```

```
1 <class 'numpy.ndarray'> (100, 785)
```




```
1 Matrix = np.mat(digits)
2 print(np.shape(Matrix))
3 plt.imshow(Matrix, cmap='Greys', interpolation='nearest')
4 plt.show()
```

```
1 (100, 785)
```




# Iterator, Enumerate, Zip

## Iterator

```python
## iterating over iterables
s = "data science"
for c in s:
    print(c, end="_")

my_iterator = iter(s) ## Generate a iterator
print("\niterator:")
print(next(my_iterator))
print(my_iterator)
print(*my_iterator)
##print(next(my_iterator)) ## Beyond Iterator 越界了
```

```
d_a_t_a_ _s_c_i_e_n_c_e_
iterator:
d
<str_iterator object at 0x7fe500c9bb50>
a t a   s c i e n c e
```

```python
## iterator over files
fid = open('cars.csv')
file_iterator = iter(fid)
#print(next(file_iterator))

while True:
    try:
        print(next(file_iterator), end="")
    except StopIteration: ## 迭代完就跳出
        break
```

```
,cars_per_cap,country,drives_right
US,809,United States,True
AUS,731,Australia,False
JAP,588,Japan,False
IN,18,India,False
RU,200,Russia,True
MOR,70,Morocco,True
EG,45,Egypt,True
```

```python
## Iterable != List
googol = iter(range(10**100))
print(next(googol))
print(next(googol))
```

```
0
1
```

## Enumerate 提取index和values

```python
my_list = ['a', 'b', 'c']
for index, value in enumerate(my_list):
    print(index, value)

for index, value in enumerate(my_list, start=10):
    print(index, value)

e = enumerate(my_list)
print(e)
print(list(e))
print(list(e)) # WHY? 迭代器，遍历一边就没了
```

```
0 a
1 b
2 c
10 a
11 b
12 c
<enumerate object at 0x7fe500cb3040>
[(0, 'a'), (1, 'b'), (2, 'c')]
[]
```

## Zip

```python
A = ['a', 'b', 'c']
B=[1, 2, 3]

C = zip(A, B)
print(list(C))

for item in C:
    print(item) # 什么都没有，因为遍历完了

print(list(C)) # WHY? 迭代器

for v1,v2 in zip(A, B):
    print(v1, v2)

print(*zip(A,B))

## Assign variables from zip
```

```
C = zip(A,B)
A1, A2 = zip(*C)
print(A1, A2)
```

```
[('a', 1), ('b', 2), ('c', 3)][]a 1b 2c 3('a', 1) ('b', 2) ('c', 3)('a', 'b', 'c')
(1, 2, 3)
```

# Numpy For Linear Algebra

## Vector in Numpy

```
import numpy as np

# 1-D array
vec = np.array([1,2,3,4]) # 数组
print(vec,vec.shape, type(vec))
```

```
[1 2 3 4] (4,) <class 'numpy.ndarray'>
```

```
# operations
print('Broadcasting: ', vec + 1) ## 向量与标量的加法运算
print('Mean: ', vec.mean())
%time print('Vectorization: ', vec**2, vec + vec)
```

```
Broadcasting:  [2 3 4 5]
Mean:  2.5
Vectorization:  [ 1  4  9 16] [2 4 6 8]
CPU times: user 564 µs, sys: 116 µs, total: 680 µs
Wall time: 695 µs
```

## Matrix in Numpy

```
# matrix class, row or col is still a matrix; MATRIX in numpy is DEPRECATED!!!
## numpy会逐渐取消对matrix的支持
mat = np.mat([[1,2],[3,4]])
print(mat)
print(mat.shape)
print(type(mat),"\n")

```

```
col = mat[:,0]
print(col)
print(col.shape)
print(type(col),"\n")

row = mat[1,:]
print(row)
print(row.shape,type(row))
```

```
[[1 2]
 [3 4]]
(2, 2)
<class 'numpy.matrix'>

[[1]
 [3]]
(2, 1)
<class 'numpy.matrix'>

[[3 4]]
(1, 2) <class 'numpy.matrix'>
```

```
# 2-D array, row or col is a 1-D array
arr2 = np.array([[1,2],[3,4]])
print(arr2)
print(arr2.shape)
print(type(arr2),'\n')

col = arr2[:,0] # 第1列
print(col)
print(col.shape)
print(type(col),'\n')

row = arr2[1,:] # 第2行
print(row)
print(row.shape)
print(type(row))
```

```
[[1 2]
 [3 4]]
(2, 2)
<class 'numpy.ndarray'>

[1 3]
(2,)
<class 'numpy.ndarray'>

[3 4]
(2,)
<class 'numpy.ndarray'>
```

## Special Matrix Generation

```
myZero = np.zeros([3,5]) # generate a 3*5 zero matrix
print(myZero,type(myZero)) # 零矩阵
```

```
[[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]] <class 'numpy.ndarray'>
```

```
myIdentity = np.identity(3) # generate a 3*3 identity matrix
print(myIdentity) # 单位矩阵
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

```
myOnes = np.ones([3,5]) # generate a 3*5 ones matrix
print(myOnes, type(myOnes))
```

```
[[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]] <class 'numpy.ndarray'>
```

```
myRand = np.random.rand(3,5) # generate a 3*5 matrix with the number in (0~1)
print(myRand) # 随机矩阵
```

```
[[0.00448604 0.51443299 0.34172675 0.52513815 0.92158036]
 [0.21892757 0.76260043 0.64660872 0.46755666 0.62721038]
 [0.86264916 0.92462231 0.4145474  0.8205234  0.24798527]]
```

```
myEyes = np.eye(3,5) # generate a 3*5 identity matrix, np.eye(3) will get a square matrix
print(myEyes)
```

```
[[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]]
```

```
print('Matrix Stack: Horizontal\n', np.hstack((myZero, myOnes))) # 水平拼接
print('Matrix Stack: Verticle\n', np.vstack((myZero, myOnes))) # 垂直拼接
```

```
Matrix Stack: Horizontal
 [[0. 0. 0. 0. 0. 1. 1. 1. 1. 1.]
 [0. 0. 0. 0. 0. 1. 1. 1. 1. 1.]
 [0. 0. 0. 0. 0. 1. 1. 1. 1. 1.]]
Matrix Stack: Verticle
 [[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]
```

## Matrix Operations

```
myOnes = np.ones([3,5]) # 中括号
myEyes = np.eye(3,5) # 小括号 # 括号不一样
print('myOnes:\n', myOnes)
print('myEyes:\n', myEyes)
```

```
myOnes:
 [[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]
myEyes:
 [[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]]
```

```
## Visit elements:
print(myEyes[1,1], myEyes[1][1]) ###### Matrix Sum
print('Sum of Matrix:myOnes+myEyes\n', myOnes + myEyes)
```

```
1.0 1.0
Sum of Matrix:myOnes+myEyes
 [[2. 1. 1. 1. 1.]
 [1. 2. 1. 1. 1.]
 [1. 1. 2. 1. 1.]]
```

```
###### sum of all elements of Matrix:sum = np.sum(*, axis =0); np.sum() = sum of all
print('myOnes:\n', myOnes)
print('Sum of elements: np.sum(myOnes)\n', np.sum(myOnes, axis=1)) # axis=1 所有列求和，压到第一行
print('Sum of each columns: sum(myOnes)\n', sum(myOnes)) # 所有行求和
print(type(myOnes), type(myEyes))
```

```
myOnes:
 [[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]
Sum of elements: np.sum(myOnes)
 [5. 5. 5.]
Sum of each columns: sum(myOnes)
 [3. 3. 3. 3. 3.]
<class 'numpy.ndarray'> <class 'numpy.ndarray'>
```

```
## Matrix Multiplication
NewMat = np.array([[1,2,3], [2, 3, 1],[1,0,0]])
print('Matrix:\n',NewMat,type(NewMat))
```

```
Matrix:
 [[1 2 3]
 [2 3 1]
 [1 0 0]] <class 'numpy.ndarray'>
```

```
print('Scalar Matrix Multiplication:\n', 10*NewMat)
```

```
Scalar Matrix Multiplication:
 [[10 20 30]
 [20 30 10]
 [10  0  0]]
```

```
print('Matrix Multiplication*:\n', NewMat*NewMat) # 矩阵元素相乘 *号开始是表示卷积
print('Matrix Multiplication@:\n', NewMat@NewMat) # 真正的矩阵相乘
```

```
Matrix Multiplication*:
 [[1 4 9]
 [4 9 1]
 [1 0 0]]
Matrix Multiplication@:
 [[ 8  8  5]
 [ 9 13  9]
 [ 1  2  3]]
```

```
print('Matrix by-Elements Multiplication: np.multiply\n', np.multiply(NewMat,
NewMat))
```

```
Matrix by-Elements Multiplication: np.multiply
 [[1 4 9]
 [4 9 1]
 [1 0 0]]
```

```
print('Matrix Elements Power: np.power\n', np.power(NewMat, 2))
```

```
Matrix Elements Power: np.power
 [[1 4 9]
 [4 9 1]
 [1 0 0]]
```

## Matrix Transpose

```
NewMat = np.array([[1,2,3],[4,5,6],[7,8,9]])
print('Matrix: \n',NewMat)
print('Matrix Transpose:\n',NewMat.T) # 转置
print('Matrix after .T: \n', NewMat) # 不会保留转置结果
print('Matrix Transpose:\n', NewMat.transpose()) ## same as the .T
print('Matrix after Transpose():\n', NewMat) # 不会保留转置结果
```

```
Matrix:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
Matrix Transpose:
 [[1 4 7]
 [2 5 8]
 [3 6 9]]
Matrix after .T:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
Matrix Transpose:
 [[1 4 7]
 [2 5 8]
 [3 6 9]]
Matrix after Transpose():
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
```

## Matrix Shape, Copy, slice, split...

```
NewMat = np.array([[1,2,3],[4, 5, 6],[7, 8, 9 ],[10,1,1]])
[m,n] = np.shape(NewMat)
print('Matrix:\n',NewMat)
print('Row and Columns:', m,n)
print('First Row:', NewMat[0]) # same as: NewMat[0,](Bad) and NewMat[0,:]
print('First Column:', NewMat.T[0])
print('Matrix Elements: ', NewMat[2,1]) ## Notice: first number is 0!!!!
print('Matrix Rows:\n', NewMat[[0,1],:]) ## first two rows
print('Matrix Colmuns:\n', NewMat[:,[1,2]]) # copy a matrix to other variable
NewMat2 = NewMat.copy() # 类比 y = x 和 z = x[:]
# compare two matrix by elements print(NewMat<NewMat2)
print('New Shape: \n', NewMat.reshape((2,6))) # 重新排列
```

```
Matrix:
```

```
 [[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10  1  1]]
Row and Columns: 4 3
First Row: [1 2 3]
First Column: [ 1  4  7 10]
Matrix Elements:  8
Matrix Rows:
 [[1 2 3]
 [4 5 6]]
Matrix Colmuns:
 [[2 3]
 [5 6]
 [8 9]
 [1 1]]
New Shape:
 [[ 1  2  3  4  5  6]
 [ 7  8  9 10  1  1]]
```

# Linear Algebra

- Determinant
- Matrix Inverse
- Matrix Rank
- Solving Linear System
- Eigen Values

## Linear Algebra

```
NewMat = np.array([[1,2,3],[4,5,6],[1,0,0]])
print("matrix:\n",NewMat)
print('Determinant:', np.linalg.det(NewMat)) # 行列式
print('Inverse Matrix:\n', np.linalg.inv(NewMat)) # 逆矩阵
print('check: inv(A)*A\n', np.linalg.inv(NewMat)@NewMat) # 验算
print('Rank:\n', np.linalg.matrix_rank(NewMat)) # 秩
b = [1,2,1]
print('Solve Linear System:\n', np.linalg.solve(NewMat, b)) # 线性方程组的结果
b2 = [1,2,1]
b2T = np.array(b2).T
print('Check by: inv(A)*b\n', np.linalg.inv(NewMat)@b2T) # 验算
```

```
matrix:
 [[1 2 3]
 [4 5 6]
 [1 0 0]]
```

```
Determinant: -3.0000000000000004
Inverse Matrix:
 [[ 0.          0.          1.        ]
 [-2.          1.         -2.        ]
 [ 1.66666667 -0.66666667  1.        ]]
check: inv(A)*A
 [[ 1.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 4.44089210e-16  1.00000000e+00  0.00000000e+00]
 [-2.22044605e-16  0.00000000e+00  1.00000000e+00]]
Rank:
 3
Solve Linear System:
 [ 1.         -2.          1.33333333]
Check by: inv(A)*b
 [ 1.         -2.          1.33333333]
```

```python
# Eigen values 特征值
Evals, Evecs = np.linalg.eig(NewMat)
print('Eigen values: \n', Evals)
print('Eigen Vectors:\n', Evecs) #NOTICE: eigenvectors are stored in columns 按列储存
## 验证：特征值*矩阵=特征向量*矩阵
print('AX , lambda*X:\n', NewMat@Evecs[:,0], Evals[0]*Evecs[:,0]) # [:,0]第一列

# use similarity matrix
sigma = Evals * np.eye(3) # 特征分解 特征值构成的矩阵 为什么是*号？答：逐行相乘
print('sigma,\n ',sigma)
# 验证
print('V*sigma*V^-1 = A:\n', Evecs@sigma@np.linalg.inv(Evecs))
```

```
Eigen values:
 [ 6.81573612 -1.1866583   0.37092218]
Eigen Vectors:
 [[-0.3482453  -0.75825177  0.19151266]
 [-0.93600993 -0.12945231 -0.8347106 ]
 [-0.05109431  0.63898072  0.51631494]]
AX , lambda*X:
 [-2.37354807 -6.37959666 -0.3482453 ] [-2.37354807 -6.37959666 -0.3482453 ]
sigma,
  [[ 6.81573612 -0.          0.        ]
 [ 0.         -1.1866583   0.        ]
 [ 0.         -0.          0.37092218]]
V*sigma*V^-1 = A:
 [[ 1.00000000e+00  2.00000000e+00  3.00000000e+00]
 [ 4.00000000e+00  5.00000000e+00  6.00000000e+00]
 [ 1.00000000e+00 -2.59536915e-16 -4.29287293e-16]]
```

```
print(np.eye(3))
print(Evals)
Evals * np.eye(3)
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
[ 6.81573612 -1.1866583   0.37092218]
```

```
array([[ 6.81573612, -0.        ,  0.        ],
       [ 0.        , -1.1866583 ,  0.        ],
       [ 0.        , -0.        ,  0.37092218]])
```

# Generate Random Number

Ref: Random https://docs.scipy.org/doc/numpy-1.10.1/reference/routines.random.html

### Generate a random float number

```
import numpy as np
np.random.seed(123) ## if set seed(123), the random number will be fixed!!!
print(np.random.rand())
print(np.random.randint(1,7))
```

```
0.6964691855978616
3
```

# Exercise

```
# 1.
NewMat = np.array([[8,1,6], [3, 5, 7],[4,9,2]])
print(NewMat)
Evals, Evecs = np.linalg.eig(NewMat.T@NewMat)
print(Evals)
print(Evecs)
```

```
[[8 1 6]
 [3 5 7]
 [4 9 2]]
[225.  12.  48.]
[[-5.77350269e-01 -7.07106781e-01  4.08248290e-01]
 [-5.77350269e-01 -1.54753999e-16 -8.16496581e-01]
 [-5.77350269e-01  7.07106781e-01  4.08248290e-01]]
```

```python
# 2.
NewMat = np.array([[8,1,6], [3, 5, 7],[4,9,2]])
print(NewMat)
print('')
U,D,V = np.linalg.svd(NewMat)
print('U:\n',U)
print('D:\n',D)
print('V:\n',V)
# 验证
print('')
tmp = U@S
print(tmp@V)
```

```
[[8 1 6]
 [3 5 7]
 [4 9 2]]

U:
 [[-5.77350269e-01  7.07106781e-01  4.08248290e-01]
 [-5.77350269e-01  5.86612687e-15 -8.16496581e-01]
 [-5.77350269e-01 -7.07106781e-01  4.08248290e-01]]
D:
 [15.          6.92820323  3.46410162]
V:
 [[-5.77350269e-01 -5.77350269e-01 -5.77350269e-01]
 [ 4.08248290e-01 -8.16496581e-01  4.08248290e-01]
 [ 7.07106781e-01 -1.15899379e-14 -7.07106781e-01]]

[[8. 1. 6.]
 [3. 5. 7.]
 [4. 9. 2.]]
```